Theoretical Computer Science 411 (2010) 2359–2367

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Computing the graph-based parallel complexity of gene assembly

Artiom Alhazov [a,b], Chang Li [a,c], Ion Petre [d,a,c,*]

[a] Department of Information Technologies, Åbo Akademi University, Finland

[b] Institute of Mathematics and Computer Science, Academy of Sciences of Moldova, Str. Academiei 5, Chişinău, MD-2028, Republic of Moldova

[c] Turku Centre for Computer Science, FIN-20520 Turku, Finland

[d] Academy of Finland, Finland

ARTICLE INFO

Article history:
Received 13 November 2007
Received in revised form 29 August 2008
Accepted 8 February 2010
Communicated by L. Kari

Keywords:
Gene assembly
Parallelism
Signed graphs
Parallel complexity
Algorithmics

ABSTRACT

We consider a graph-theoretical formalization of the process of gene assembly in ciliates introduced in Ehrenfeucht et al. (2003) [3], where a gene is modeled as a signed graph. The gene assembly, based on three types of operations only, is then modeled as a graph reduction process (to the empty graph). Motivated by the robustness of the gene assembly process, the notions of parallel reduction and parallel complexity of signed graphs have been considered in Harju et al. (2006) [7]. We describe in this paper an exact algorithm for computing the parallel complexity of a given signed graph and for finding an optimal parallel reduction for it. Checking the parallel applicability of a given set of operations and scanning all possible selections amount to a high computational complexity. We also briefly discuss a faster approximate algorithm that however, cannot guarantee finding the optimal reduction.

1. Introduction

Ciliates are an old and diverse group of unicellular eukaryotes. One of their unique features is that they have two types of functionally different nuclei: micronuclei and macronuclei. Micronuclei are the germline nuclei, where no transcription takes place, while the macronuclei are the somatic nuclei. The difference between the micronuclear and the macronuclear genome is striking, especially in stichotrichs, on which we concentrate in the following. Thus, while macronuclear genes are contiguous sequences placed in general on their own molecules, micronuclear genes are placed on long chromosomes, interrupted by stretches of non-coding material. Even more striking is the fact that the micronuclear genes are split into coding blocks (called MDSs, up to 44 of them in certain species), with the blocks arranged in a scrambled order, separated by non-coding material. Some blocks may even be inverted! At some stage during conjugation, ciliates assemble the blocks in the orthodox order to yield the macronuclear gene. The assembly process is facilitated by the structure of the MDSs: each coding block ends with a short sequence of nucleotides that is repeated at the beginning of the coding block that should follow it in the orthodox order. This structure reminds us of the linked lists data structure in Computer Science. We refer to [16,12] for more details on ciliates and on gene assembly.

Two mathematical models were introduced for gene assembly: an intermolecular model, see [13,14] and an intramolecular model, see [6,17]. The intramolecular model, that we consider in this paper, has been formalized on several levels of abstraction, including permutations, strings, and graphs, see [11] for details and [4] for a monograph on the mathematical theory of gene assembly. In the following we will only consider a formalization of genes through graphs.

* Corresponding author at: Department of Information Technologies, Åbo Akademi University, Finland. Tel.: +358 22153361.
E-mail addresses: artiom@math.md (A. Alhazov), lchang@abo.fi (C. Li), ion.petre@abo.fi (I. Petre).

0304-3975/$ – see front matter
doi:10.1016/j.tcs.2010.02.007

Table 1
The order (number of vertices) n of the smallest graphs of complexity c, see examples in [8].

c	1	2	3	4	5	6
n	1	2	3	5	12	24

A micronuclear gene may be represented through the (scrambled) sequences of its coding blocks. Each coding block in turn, may be represented by its pair of left and right pointers. Thus, the block B_i that should come on the ith position in the orthodox order, will be denoted as $B_i = i(i+1)$. The first block begins with a specific marker that we may ignore in our abstraction and denote $B_1 = 2$. Similarly, the last block will be denoted as $B_n = n$. If a block B_i comes inverted in the micronuclear gene, then it will be denoted as $\overline{B_i} = (\overline{i+1})\overline{i}$. As such, a micronuclear gene may be denoted as a *signed double occurrence string*, see [3] for more details. On a different level of abstraction, one may replace the signed double occurrence string with its correspondent signed overlap graph. For a micronuclear gene consisting of n coding blocks, $n \geq 1$, and having the associated string u, its corresponding signed overlap graph will have the form $G_u = (\{2, \ldots, n\}, E_u, \sigma_u)$, where

- $ij \in E_u$ if and only if i and j overlap in u, i.e., $u = u_1 i' u_2 j' u_3 i'' u_4 j'' u_5$ or $u = u_1 j' u_2 i' u_3 j'' u_4 i'' u_5$, for some strings u_i, $1 \leq i \leq 5$, where $i', i'' \in \{i, \bar{i}\}$ and $j', j'' \in \{j, \bar{j}\}$
- $\sigma(i) = +$ if both i and $\bar{i}$ occur in u, and $\sigma(i) = -$ otherwise.

The intramolecular model for gene assembly introduced in [6,17] consists of three molecular operations, whose description we omit here. It is enough for the purpose of this paper to mention that each operation combines two or, in one case, even three, coding blocks into a bigger coding block by splicing on their common pointers. Since a pointer is only important for gene assembly when it is placed at the extremity of a coding block (and not when inside a bigger coding block), it follows that the process of gene assembly may be thought of as a process of removing pointers. Note also that the final assembled gene is a contiguous sequences of nucleotides containing no more pointers. Based on these observations, the molecular operations of [6,17] may be formalized as rewriting rules for signed double occurrence strings as well as rewriting rules for signed overlap graphs. It may be proved mathematically that as far as gene assembly is concerned, both formalizations are equivalent, with an assembled gene corresponding to the empty string and to the empty graph, see [4] for details.

Given the crucial role that parallelism plays in biochemical processes, it is important to consider a notion of parallelism in the graph-based mathematical framework for gene assembly. Indeed, parallelism in this context has been defined in [7] as follows: a set S of operations can be applied in parallel to a graph G if all sequential compositions of operations in S are applicable to G. It is proved in [7] that in this case, all sequential compositions of operations in S lead to the same result when applied to G. This leads to considering parallel reduction strategies for a given graph G, where one applies a number of operations in parallel to G, obtaining a new graph G' and continues doing so until obtaining the empty graph. Furthermore, this yields a measure of complexity for a signed graph in terms of the minimal number of parallel steps needed to reduce the graph to the empty one. This measure of complexity may be related to the degree of complexity of the process of gene assembly. It has been shown in [8] that all known ciliate genes may be assembled in at most two parallel steps. Graphs of higher complexity are known, as summarized in Table 1, based on examples of [8].

A number of partial results have been obtained on whether an upper bound exists on the parallel complexity of signed graphs, see [7,10,8], but the problem in its full generality remains open.

Open problem: *Is the parallel complexity of signed graphs finitely bounded?*

Even (seemingly) simpler variants of the problem, where the question is asked for signed trees or for unsigned graphs, remain open up to date. It has only been shown that the parallel complexity of negative trees is at most two, while that of positive trees is at most three, see [9]. On the other hand, examples of signed trees of parallel complexity five and examples of signed graphs of parallel complexity six are known, see [8]. The difficulty of the problem is perhaps best illustrated by the fact that no efficient decision procedure is known for whether a given set of operations S is applicable to a given graph G. The only currently known decision procedure is based on the very definition of parallelism and it involves checking the applicability of all sequential compositions of the operations, a tedious procedure even for small sets of operations. It is not difficult to prove, see [1], that the problem is in the class of complexity coNP.

Computing the parallel complexity of a given gene/graph is even more involved: not only that sets of operations must be applied in parallel one after another, but also an optimization problem in terms of minimizing the number of parallel steps needed to reduce the graph, must be solved. It has been proved in [1] that for a given graph G and sets of operations $S_1, \ldots, S_n$, deciding whether $S_n \circ \cdots \circ S_1$ is a parallel reduction of G is a P^{NP} problem. Moreover, deciding whether the complexity of G is at most k, for a given $k \geq 1$, is an NP^{NP} problem.

We describe in this paper an algorithm to compute the parallel complexity of a given signed graph and to find in the same time an optimal parallel reduction for it. Given the current gaps in the theory of parallel complexity, the algorithm is essentially an exhaustive search. Consequently, it is not surprising that despite several cut-offs in the search algorithm, its complexity remains huge. We give it here an upper bound on the scale of $O\left(n^{2n+4}/d^n\right)$, where d is a constant and n is the number of nodes. It is an open problem whether a faster algorithm can be given based on the current theory of gene assembly. We also describe an approximate, Greedy type of algorithm that cannot guarantee finding the optimal parallel reduction (while its complexity remains highly exponential). An implementation of the exact algorithm, as well as a Greedy-style approximation algorithm are available in [15].

2. Preliminaries

A signed graph is a triple $G = (V, E, \sigma)$, where $G = (V, E)$ is an undirected graph and $\sigma : V \to \{+, -\}$. Edges between $u, v \in V$ are denoted by uv ($uv = vu$). We let $V^+ = \sigma^{-1}(+)$ and $V^- = \sigma^{-1}(-)$. By $N_G(u) = \{v \in V \mid uv \in E\}$ we denote the neighborhood of $u \in V$.

For signed graphs $G_1 = (V_1, E_1, \sigma_1)$ and $G_2 = (V_2, E_2, \sigma_2)$, we will need the following graph-theoretic operations:

- If $V_1 \cap V_2 = \emptyset$, then we denote $G_1 \cup G_2 = (V, E, \sigma)$, $V = V_1 \cup V_2$, $E = E_1 \cup E_2$, $\sigma = \sigma_1 \cup \sigma_2$;
- $G_1 \setminus G_2 = (V_1, E_1 \setminus E_2, \sigma_1)$;
- $G_1 \Delta G_2 = (G_1 \setminus G_2) \cup (G_2 \setminus G_1)$.

For a set $S \subseteq V$ we denote by $G|_S = (S, E \cap (S \times S), \sigma|_S)$ the subgraph induced by S. We also write $G - S = G|_{V \setminus S}$. For a set $S \subseteq V$ we denote by $K_G(S) = (S, \{uv \mid u, v \in S,\ u \neq v\}, \sigma|_S)$ the clique induced by S. For sets $S_1, S_2 \subseteq V$ with $S_1 \cap S_2 = \emptyset$, we use the notation $K_G(S_1, S_2) = (S, \{uv \mid u \in S_1,\ v \in S_2\}, \sigma|_{S_1 \cup S_2})$ to represent the complete bipartite graph induced by S_1 and S_2.

For a graph $G = (V, E, \sigma)$, we denote $neg(G) = (V, E, \sigma')$, where $\sigma'(u) = -$ if and only if $\sigma(u) = +$, $u \in V$ the graph obtained from G by complementing its signing. Then $com(G) = neg(K_G(V) \setminus G)$ denotes the graph obtained from G by complementing both its edges and its signing. For a set $S \subseteq V$, we denote $com_S(G) = com(G|_S) \cup (G \setminus K_G(S))$. Finally, for a node $u \in V$ we denote by $loc_u(G) = com_{N_G(u)}(G)$ the graph with complemented edges and signing over the neighborhood of u.

3. A graph-based model for gene assembly

We recall in this section the graph-based mathematical framework for gene assembly, as introduced in [3], see also [4].

Definition 1. Consider a signed graph $G = (V, E, \sigma)$.

- For $x \in V^-$, if $N_G(x) = \emptyset$, then the *graph negative rule* gnr is applicable to x and $\mathsf{gnr}_x(G) = G - \{x\}$.
- For $x \in V^+$, the *graph positive rule* gpr is applicable to x and $\mathsf{gpr}_x(G) = loc_x(G) - \{x\}$.
- For $xy \in V^- \times V^-$, the graph double rule gdr is applicable to x, y and $\mathsf{gdr}_{x,y}(G) = (G \setminus \{x, y\}, E', \sigma|_{V \setminus \{x,y\}})$, where E' is obtained from E by complementing the edges that join nodes in $N_G(x)$ to nodes in $N_G(y)$. Thus, for $p, q \in V \setminus \{x, y\}$, the edge relationship between p and q will change if and only if

$$\begin{aligned} &p \in N_G(x) \setminus N_G(y), \quad \text{and} \quad q \in N_G(y)\\ &p \in N_G(y) \setminus N_G(x), \quad \text{and} \quad q \in N_G(x)\\ &p \in N_G(x) \cap N_G(y), \quad \text{and} \quad q \in N_G(x) \Delta N_G(y) \end{aligned}$$

Example 1. Applications of a gpr operation and a gdr operation are illustrated in Fig. 1.

The sets of all gnr, gpr and gdr operations are denoted by GNR, GPR and GDR, respectively. We also use notations $dom(\mathsf{gnr}_x) = \{x\}$, $dom(\mathsf{gpr}_x) = \{x\}$ and $dom(\mathsf{gdr}_{x,y}) = \{x, y\}$. We extend the notation to sets: $dom(S) = \cup_{r \in S} dom(r)$.

For an operation r and a sequential composition φ of operations, we say that $\varphi \circ r$ is applicable to G if r is applicable to G and φ is applicable to $r(G)$ (clearly, $\emptyset$ is always applicable to G).

Consider $\varphi = r_k \circ \cdots \circ r_1$ where $r_1, \ldots, r_k \in \mathrm{GNR} \cup \mathrm{GPR} \cup \mathrm{GDR}$. Whenever φ is applicable to G, we define the predicate $applicable_\varphi(G)$ as true, and we naturally denote the result by $\varphi(G) = r_k(\ldots(r_1(G))\ldots)$.

4. Parallelism

We recall in this section the notion of parallelism for the reduction of signed graphs, as introduced in [7].

Definition 2. We say that operations $S \subseteq \mathrm{GNR} \cup \mathrm{GPR} \cup \mathrm{GDR}$ are applicable in parallel to G if any sequential composition of the operations in S is applicable to G.

We recall the following lemma (Theorem 6.4 from [7]).

Lemma 1 ([7]). *If S is applicable in parallel to G, then for any two sequential compositions φ_1, φ_2 of the operations in S, we have $\varphi_1(G) = \varphi_2(G)$.*

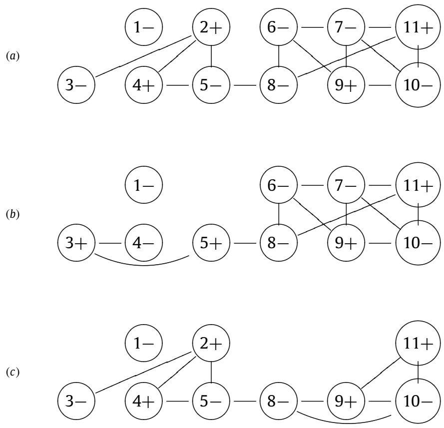

Fig. 1. Graphs (a) G, (b) $\mathsf{gpr}_2(G)$ and (c) $\mathsf{gdr}_{6,7}(G)$.

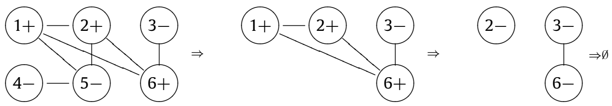

Fig. 2. A graph G_6 with 6 nodes and parallel complexity 3. Its reduction corresponding to an optimal strategy $\{\mathsf{gnr}_2, \mathsf{gdr}_{3,6}\} \circ \{\mathsf{gpr}_1\} \circ \{\mathsf{gdr}_{4,5}\}$.

Therefore, whenever S is applicable in parallel to G, we may consider the graph $S(G)$ obtained by applying S to G as the result of applying the operations of S to G in an arbitrary order.

We now recall the definition of parallel complexity of signed graphs.

Definition 3 (*[10]*). Let G be a signed graph. For sets $S_1, \ldots, S_k \subseteq \mathsf{GNR} \cup \mathsf{GPR} \cup \mathsf{GDR}$, we say that $R = S_k \circ \cdots \circ S_1$ is applicable to G if S_i is applicable to $(S_{i-1} \circ \cdots \circ S_1)(G)$, for all $1 \le i \le k$. If $R(G) = \emptyset$, then we say that R is a parallel reduction strategy for G. We say that the parallel complexity of R is $\mathcal{C}(S_k \circ \cdots \circ S_1) = k$. The *parallel complexity* $\mathcal{C}(G)$ of G is defined as

$$\mathcal{C}(G) = \min\{\mathcal{C}(R) \mid R \text{ is a parallel reduction strategy for } G\}.$$

A parallel reduction strategy R for a graph G is called *optimal* if $\mathcal{C}(R) = \mathcal{C}(G)$.

Example 2. An example of a graph, its parallel complexity and an optimal strategy to reduce it in parallel to the empty graph are given in Fig. 2.

Consider a signed graph $G = (V, E, \sigma)$ and sets $V_n \subseteq V^- \cap \{u \in V \mid N_G(u) = \emptyset\}$, $V_p \subseteq V^+$ and $E_d \subseteq E \cap (V^- \times V^-)$. Consider the following problem: decide whether a set $S = \{\mathsf{gnr}_x \mid x \in V_n\} \cup \{\mathsf{gpr}_x \mid x \in V_p\} \cup \{\mathsf{gdr}_{x,y} \mid xy \in V_d\}$ of operations is applicable to G in parallel. The following lemma summarizes the current partial solution to the problem.

Lemma 2 (*[7]*). *Consider a signed graph G and a set $S \subset \mathsf{GNR} \cup \mathsf{GPR} \cup \mathsf{GDR}$ of operations applicable to G. Then S is applicable in parallel to G if and only if $N_{G|_{dom(S)}}(u) = \emptyset$ for all $\mathsf{gpr}_u \in S$ and $S \cap \mathsf{GDR}$ is applicable in parallel to G.*

The major problem when searching for the parallel complexity is that there exists no efficient decision procedure for whether or not a set of operations is applicable in parallel to a given graph. Consequently, to solve this decision problem, one has to check that all sequential compositions of the operations are applicable to the given graph. This will contribute essentially to the high computational complexity of the algorithm described in the next section.

5. An algorithmic approach

Clearly, every graph can be successfully reduced to the empty graph since applying any operation decreases the number of nodes, and an operation can be applied to any non-empty graph. We are faced here however with a problem on two levels. On one hand, we are looking for parallel strategies, and thus, we have to decide whether or not the operations considered in every step may indeed be applied in parallel. On the other hand, we have an optimization problem: we look for a parallel strategy of minimal length in terms of the number of parallel steps.

The main observation behind our approach is the following:

$$\mathcal{C}(G) = 1 + \min\{\mathcal{C}(G') \mid G' = S(G),\ S \subseteq \mathsf{GNR} \cup \mathsf{GPR} \cup \mathsf{GDR}\}. \tag{1}$$

As such, our exact algorithm will make an exhaustive search through all sets S applicable in parallel to G. Note that taking a Greedy approach here and considering only *largest* sets S may not lead to the optimum, see [8] and the discussion in Section 8.

An immediate improvement (limiting considered sets of operations) is that the isolated negative nodes can always be reduced (i.e., corresponding gnr operations can be applied) as soon as they become isolated.

Another improvement (limiting the considered permutations) is possible based on Lemma 2: only permutations of gdr operations have to be considered. In addition, we only check that the positive nodes to be reduced are not adjacent to any other selected nodes. Note that constructing the sets of operations applicable in parallel is incremental; therefore, checking parallel applicability of an operation set is reduced to checking the "compatibility" of the last operation with the other ones already in the set.

Yet another improvement (limiting graphs actually considered) is comparing the number of steps needed to reduce the current graph with the best complexity candidate found so far: if the current graph cannot improve the current optimum, then we cut the search tree.

It is important to observe that there are three main levels of complexity in this optimization problem:

- Considering different sequential compositions of the operations from a set of operations for checking its applicability in parallel: for k operations, $k!$ sequential compositions exist.
- Scanning all sets of operations applicable in parallel to the given graph: for a graph with n nodes, there may be more than 2^n sets of operations applicable in parallel.
- Reducing the complexity of a graph to the complexity of smaller graphs: not only do we need to determine all sets of operations applicable in parallel to G, but also to perform related search for all resulting graphs G' and the graphs obtained by further reductions. The number of graphs to be examined is non-polynomial.

6. An exhaustive search algorithm

We now present an algorithm for finding the parallel complexity and an associated strategy. The answer is obtained by calling the function `Complexity`, giving it as parameters the corresponding graph, the empty set and the number of nodes plus one.

Throughout this section $G = (V, E, \sigma)$ is a signed graph, with $V = \{1, \dots, n\}$. We need the notation $Seq(S)$ for the set of all sequential compositions of elements from $S = \{r_j \mid 1 \le j \le k\} \subseteq \mathsf{GNR} \cup \mathsf{GPR} \cup \mathsf{GDR}$:

$$Seq(S) = \{r_{i(k)} \circ \cdots \circ r_{i(1)} \mid i : \{1 \dots k\} \to \{1 \dots k\} \text{ bijection}\}.$$

In order to consider the sets of operations just once, we add the operations according to the order $>$ defined on the set of operations applicable to the current graph, which is induced by the order of nodes; note that operations of different types cannot be applicable to the same node. For any $p, q, r, s, t \in V$ we define this order relation as follows:

- $\mathsf{gpr}_p > \mathsf{gnr}_q$,
- $\mathsf{gdr}_{p,q} > \mathsf{gnr}_s$,
- $\mathsf{gnr}_p > \mathsf{gnr}_q$, if $p > q$,
- $\mathsf{gpr}_p > \mathsf{gpr}_q$, if $p > q$,
- $\mathsf{gpr}_p > \mathsf{gdr}_{q,s}$, if $p > \min(q, s)$,
- $\mathsf{gdr}_{p,q} > \mathsf{gpr}_s$, if $\min(p, q) > s$,
- $\mathsf{gdr}_{p,q} > \mathsf{gdr}_{s,t}$, if either $\min(p, q) > \min(s, t)$, or $\min(p, q) = \min(s, t)$ and $\max(p, q) > \max(s, t)$.

We use the extension of this relation by writing $r > S$, where S is a set of operations, if $r > s$ for all $s \in S$. Notice that since the operations are added to the sets according to the order $>$, in the implementation of the algorithm where sets are represented by arrays or lists, $r > S$ is equivalent to $r > s$, where s is the last element added to S.

We are now ready to explain the central function of the algorithm. Function `Complexity` takes three parameters: a graph G, a set S of operations already chosen to be applied in the current step, and an integer *bound*, and returns the best reduction strategy of G in less than *bound* steps, with the first step of the reduction starting with S.

The first part of the function has two cases. If some operation was already chosen ($S \neq \emptyset$), then the `Complexity` function is called recursively to check the case when the current step is finished ($G \leftarrow S(G), S \leftarrow \emptyset, bound \leftarrow bound - 1$). If the complexity returned plus one is smaller than *bound*, then the corresponding solution (S in the current step and the returned strategy) becomes the current best solution, and *bound* is decreased accordingly.

If no operation was yet chosen for this step ($S = \emptyset$), then all possible gnr operations are chosen to be applied (i.e., added to S), removing isolated negative nodes.

The second part of the function again has two cases. If applying the chosen operations to G yields an empty graph, then the function returns 0 if $S = \emptyset$, or 1 otherwise.

If $S(G) \neq \emptyset$, then the function should consider extending the set of operations chosen in the current step. This is done in the following way: starting from the first operation possible if $S = \emptyset$ or with the next operation after the last one added to S, for every operation r (according the order "$>$") the function checks whether r can be applied in parallel with S to G. In

```
Input: graph G, set S, integer bound
Output: integer, strategy
Data: strategy R, R'; integer i
R ← ∅;
if S = ∅ then
 | S ← {gnr_u | u ∈ V⁻, N_G(u) = ∅};
else if bound > 1 then
 | (R', i) ←Complexity(S(G), ∅, bound − 1);
 | if i + 1 < bound then
 | | bound ← i + 1;
 | | R ← R' ∘ S;
 | end
end
if S(G) = ∅ then
 | if S = ∅ then
 | | return (0, ∅);
 | else
 | | return (1, S);
 | end
else
 | foreach r > S do
 | | if Check(G, S, r) then
 | | | (R', i) ←Complexity(G, S ∪ {r}, bound);
 | | | if i < bound then
 | | | | bound ← i;
 | | | | R ← R';
 | | | end
 | | end
 | end
end
return (R, bound);
```

Function Complexity. The central routine: find the best reduction strategy of G in less than *bound* steps, with the first-step reduction starting with S. If the current set is empty, perform possible gnr reductions, otherwise consider the reduced graph for the next step, unless the current graph cannot improve the current optimum. If the reduced graph is empty, return 0 or 1, otherwise for every operation r following the current set S according to order $>$, if r is applicable in parallel together with S, add r to S and call the same function. Choose the best value.

case it can, the function Complexity is called recursively to check the case when r is also chosen in this step ($S \leftarrow S \cup \{r\}$). If the complexity returned is smaller than *bound*, then the returned strategy is the current best solution.

In the end, the current best strategy and its length are returned.

```
Input: graph G, set S, op r
Output: boolean
Data: composition φ
if r ∈ GPR then
 | return N_G(dom(r)) ∩ dom(S) = ∅;
else
 | foreach φ ∈ Seq({S ∪ r} ∩ GDR) do
 | | if not(applicable_φ(G)) then
 | | | return false;
 | | end
 | end
 | return true;
end
```

Function Check. Is the operation r applicable in parallel with the set S of operations? Based on Lemma 2.

Finally, we explain the function Check. It considers a graph G, a set S of operations applicable in parallel to G, and an operation r, and decided whether r is applicable in parallel with S to r. Based on Lemma 2, if $r \in$ GPR, then the function returns true if and only if the node associated to r is not adjacent to any of the nodes associated to operations from S; if $r \in$ GDR, then the function returns true if and only if any sequential composition of GDR operations in $S \cup \{r\}$ can be applied to G.

7. Complexity analysis of the algorithm

A signed graph with n nodes can be represented in space $O(n^2)$ by an adjacency matrix $(m_{i,j})_{1\le i\le n, 1\le j\le n}$ and a sign vector $(s_i)_{1\le i\le n}$. For a depth-first exploring the space of possible reductions it is enough to remember the graphs obtained at various stages of the reduction; since any graph is reduced in at most n steps, space $O(n^3)$ suffices. From the further explanation it will be clear that additional data used for the computation does not increase the cubic order. Note that the intermediate graphs could be recovered from the strategy (the list of operations applied), giving the time complexity $O(n^2)$, but recomputing them is not practical since the bottleneck is time, not space. In what follows we speak about time complexity.

Removing isolated negative nodes can be done in $O(n^2)$. Applying one gpr or gdr operation can be done in $O(n^2)$. Testing applicability of k gdr operations in any order can be done in $O(k! \cdot kn^2)$; this is also the time complexity of `Check` routine.

Lemma 3. *The number of sets of operations tested for applicability in parallel to any graph G with n nodes does not exceed some function in $O(n^{n/2} \cdot c^n)$ for $c = 2/\sqrt{e}$.*

Proof. Consider a graph $G = (V, E)$. Let us denote by $s(G)$ the exact number of operation sets tested for applicability in parallel to G. Let V_1 be the set of all positive and negative isolated nodes in G, let $V_2 = V \setminus V_1$, $|V_1| = p$ and $|V_2| = n - p$. Clearly, $s(G) \le s(G|_{V_1}) \cdot s(G|_{V_2})$: any possible set is a union of possible sets of the positive and negative parts. It is also easy to see that $s(G|_{V_1}) \le 2^p$ and the estimate is exact when, i.e., $G|_{V_1} = (V_1, \emptyset)$: the number of possible sets of operations for a graph with positive and/or isolated nodes does not exceed the number of subsets of its nodes, and they are equal when all nodes are isolated.

Now consider the rest of the graph, where only gdr operations may be applied, with $n' = n - p$ nodes. The first operation can be chosen in at most $n'(n'-1)/2$ ways, the ith operation can be chosen in $(n'-2i)(n'-2i-1)/2$ ways. Using the multinomial formula, this gives us at most $\frac{n'!}{(n'-2k)!2^k}$ sequences of pairs of nodes chosen from n' nodes. The corresponding number of *sets* is obtained ignoring the order of pairs, i.e., dividing this number by $k!$. Since up to $n/2$ node pairs can be selected, the total estimate is

$$t(n') = \sum_{k=0}^{n'/2} \frac{n'!}{(n'-2k)!k!2^k}, \tag{2}$$

and $s(G|_{V_2}) \le t(n')$. The latter is an equality, e.g., when V_2 is complete.

Notice that the last ($k = \lfloor n'/2 \rfloor$) term of $t(n')$ is equal to the product of odd numbers not exceeding n',

$$prodd(n') = 1 \cdot 3 \ldots (2\lceil n'/2\rceil - 1),$$

which alone grows faster than $2^{n'}$. Thus, for sufficiently large n, for any graph G with n nodes, out of which p nodes are positive,

$$s(G) \le s(G|_{V_1}) \cdot s(G|_{V_2} \le 2^p \cdot t(n-p) \le t(n) = s(K_n^-),$$

where K_n^- is a complete negative graph. Hence, it suffices to prove the lemma for complete negative graphs.

Let us rewrite (2) in the following way:

$$t(n') = \sum_{k=0}^{n/2} \frac{n!}{(n-2k)!(2k)!} \cdot \frac{(2k)!}{k!2^k} = \sum_{k=0}^{n/2} C_{2k}^n \cdot prodd(2k) \le 2^n \cdot prodd(2\lceil n/2\rceil).$$

From Stirling's formula, we get $n! = \Theta(\sqrt{n}(n/e)^n)$. In case when n is even, $t(n) \le 2^n \cdot prodd(n)$ and $prodd(n) = \frac{n!}{(n/2)!2^{n/2}} \in O((n/e)^{n/2})$. In case when n is odd, $t(n) \le 2^n \cdot prodd(n-1)$ and $prodd(n-1)$ is also $O((n/e)^{n/2})$. Therefore, $t(n) \in O(n^{n/2} \cdot (2/\sqrt{e})^n)$, which proves the lemma. □

Lemma 4. *The number of times the `Complexity` function is recursively called does not exceed $n! \cdot 2^n$.*

Proof. A sequential composition of operations can be written as a sequence of nodes on which the operations are applied; the operations can be recovered from the context. Therefore, the total number of possibilities is bounded by $n!$.

Unless two subsequent nodes correspond to a gdr operation, we might also need to know if the associated operations are applied in the same step or not. In this way, there may be at most 2^{n-1} ways to partition a sequential composition of operations into a parallel strategy. Hence, the total number of times we may arrive to the empty graph does not exceed $1/2 \cdot n! \cdot 2^n$.

By a similar argument, the number of sub-strategies reducing k nodes is at most $1/2 \cdot n!/(n-k)! \cdot 2^k$. Summing up over $0 \le k \le n$,

$$\sum_{k=0}^{n} \frac{1}{2} \cdot \frac{n!}{(n-k)!} \cdot 2^k \le n! \cdot \frac{1}{2} \sum_{k=0}^{n} 2^k \le n! \cdot 2^n. \quad \square$$

Theorem 1. *The present algorithm is in $O\left(\frac{n^{2n+4}}{d^n}\right)$ for $d = e^2/\sqrt{8}$.*

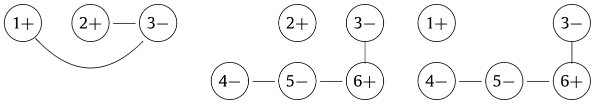

Fig. 3. Three options for the first step of a Greedy type reduction of the graph G_6 in Fig. 2 (choose as many operations as possible); all three graphs are reducible in 3 steps, yielding 4 as a Greedy-computed parallel complexity for G_6, which is not optimal.

Proof (*Sketch*). The principal term of the algorithm's complexity comes from checking the parallel applicability of different sets of operations for different intermediate graphs by applying them in different order. This gives us the time complexity of $O(\lfloor n/2\rfloor! \cdot n^3) \cdot O(n^{n/2} \cdot c^n)) \cdot O(n! \cdot 2^n)$ for $c = 2/\sqrt{e}$. According to Stirling's formula for $n!$, $n! \in \Theta(\sqrt{n}(n/e)^n)$, so we can rewrite the time complexity of the algorithm as

$$O\left(\frac{n^{n/2} \cdot n^{7/2}}{2^{n/2} \cdot e^{n/2}} \cdot \frac{n^{n/2} \cdot 2^n}{e^{n/2}} \cdot \frac{n^n \cdot n^{1/2} \cdot 2^n}{e^n}\right).$$

Simplifying this expression, we obtain the estimate in the theorem statement. □

8. Heuristics

Instead of top–down recursive examination of the graph space one could consider a bottom–up approach: compute the parallel complexity for all graphs of size k, from $k = 0$ to $k = n - 1$, by only examining one-step reductions and consulting the complexity already computed for the smaller graphs. However, the number of possible signed graphs of size n is $2^{n(n+1)/2}$, so the space requirements becomes prohibitive as n grows. Even computing it only for the graphs obtained in the reduction process of the given graph (caching) leads to hitting the space barrier before the time barrier.

Since the known complexity is so high, a natural question arises whether heuristic algorithms can help. E.g., let us consider the Greedy approach: examine only *maximal* sets of operations in each step of the reduction. In other words, we only consider reducing $\mathcal{C}(G)$ to $\mathcal{C}(G')$ if G' is obtained from G by a parallel application of the operations in set S, where no set $S' \supsetneqq S$ is applicable in parallel to G. Based on the same arguments used in analyzing the computational complexity of our algorithm, it may be seen that this Greedy approach remains highly exponential, since it embeds the same three main levels of complexity discussed in Section 5. Regarding the output that the Greedy algorithm yields, in many example we have computed, this approach finds the optimum. However, one can see from the example in Fig. 2 that, since operations $\mathsf{gdr}_{4,5}$ and gpr_6 are applicable in parallel to G_6, the Greedy approach does not examine $\{\mathsf{gdr}_{4,5}\}$ because it is not a maximal set. The maximal sets are $\{\mathsf{gdr}_{4,5}, \mathsf{gpr}_6\}$, $\{\mathsf{gpr}_1\}$ and $\{\mathsf{gpr}_2\}$. It turns out that the optimal strategies are missed in this case, see Fig. 3.

9. Discussion

The concept of parallelism in ciliate gene assembly leads to some challenging problems in graph theory. E.g., deciding whether a given set of operations is applicable in parallel to a given graph is a coNP problem, while deciding whether a given composition of some sets of operations is a parallel reduction of a given graph is a P^{NP} problem, see [1]. As such, it is not surprising that the exact algorithm discussed in this paper is essentially an exhaustive search through all possible parallel reductions of the given graph. The question whether faster exact algorithms exist and thus, whether the decision problems above lay in smaller complexity classes, remains open. The fact that an approximate Greedy algorithm also has high complexity suggests that answering this question may be very difficult.

The problem that we consider in this paper, of finding the shortest parallel reduction for a given graph is motivated by the gene assembly process: the operations that we consider are the mathematical formulations of a molecular model for gene assembly. However, the high complexity of the exact solution we give to the problem has no implications with respect to the biological relevance of the intramolecular model for gene assembly in ciliates. The ciliates need not solve the optimization problem we have considered in this paper and most likely, need not scan all reduction strategies for their genes, as we do in our algorithm. Any reduction strategy yields an assembled gene, see [5], and thus, as far as gene assembly is concerned, it is just as good as the optimal parallel reduction that our algorithm strives to find. As far as parallel strategies are concerned, it is easy to verify based on [2,15] that all graphs corresponding to the currently known gene patterns may be reduced in at most two parallel steps.

An implementation of the algorithm reported in this paper can be found in [15]. In fact, that implementation does not use recursion, but rather is done by backtracking, a non-recursive depth-first search of the strategy tree with cuts. The implementation in [15] also contains a heuristic, Greedy-style approach, slightly different than the one presented in this paper. The main difference consists in considering the gnr, gpr, gdr operations in a certain fixed order. Even though in each step the algorithm adds as many operations as possible to the current reduction set, the final set may not be maximal in the sense of Section 8. E.g., an operation that could have been added to the current set may not have been considered because it was not in the scanning range of the current step. Nevertheless, all maximal sets of operations are eventually considered in the algorithm. Thus, this version is more thorough than the Greedy algorithm described in Section 8, but it still fails to always find an optimal strategy (and is of higher complexity).

Acknowledgements

A.A. gratefully acknowledges the support by Academy of Finland, project 203667 and by the Science and Technology Center in Ukraine, project 4032. C.L. gratefully acknowledges the support by Academy of Finland, project 203667. I.P. gratefully acknowledges the support by Academy of Finland, project 108421.

References

[1] A. Alhazov, I. Petre, V. Rogojin, Computing the parallel complexity of signed graphs: An improved algorithm, Theoret. Comput. Sci. 410 (24–25) (2009) 2308–2315.
[2] A. Cavalcanti, T.H. Clarke, L. Landweber, MDS_IES_DB: A database of macronuclear and micronuclear genes in spirotrichous ciliates, Nucl. Acids Res. 33 (2005) 396–398.
[3] A. Ehrenfeucht, T. Harju, I. Petre, D.M. Prescott, G. Rozenberg, Formal systems for gene assembly in ciliates, Theoret. Comput. Sci. 292 (2003) 199–219.
[4] A. Ehrenfeucht, T. Harju, I. Petre, D.M. Prescott, G. Rozenberg, Computation in Living Cells: Gene Assembly in Ciliates, Springer, 2003.
[5] A. Ehrenfeucht, I. Petre, D.M. Prescott, G. Rozenberg, Universal and simple operations for gene assembly in ciliates, in: V. Mitrana, C. Martin-Vide (Eds.), Words, Sequences, Languages: Where Computer Science, Biology and Linguistics Meet, Kluwer Academic, Dortrecht, 2001, pp. 329–342.
[6] A. Ehrenfeucht, D.M. Prescott, G. Rozenberg, Computational aspects of gene (un)scrambling in ciliates, in: L.F. Landweber, E. Winfree (Eds.), Evolution as Computation, Springer, Berlin, Heidelberg, New York, 2001, pp. 216–256.
[7] T. Harju, C. Li, I. Petre, G. Rozenberg, Parallelism in gene assembly, Nat. Comput. 5 (2) (2006) 203–223.
[8] T. Harju, C. Li, I. Petre, Parallel complexity of signed graphs for gene assembly in ciliates, Soft Computing — A Fusion of Foundations Methodol. Appl. 12 (8) (2008) 731–737.
[9] T. Harju, C. Li, I. Petre, Graph theoretic approach to parallel gene assembly, Discrete Appl. Math. 156 (18) (2008) 3416–3429.
[10] T. Harju, C. Li, I. Petre, G. Rozenberg, Complexity measures for gene assembly, in: K. Tuyls (Ed.), Proceedings of the Knowledge Discovery and Emergent Complexity in Bioinformatics Workshop, in: Lecture Notes in Bioinformatics, vol. 4366, Springer, 2007.
[11] T. Harju, I. Petre, G. Rozenberg, Gene assembly in ciliates: Formal frameworks, Bull. EATCS 82 (2004) 227–241.
[12] C.L. Jahn, L.A. Klobutcher, Genome remodeling in ciliated protozoa, Ann. Rev. Microbiol. 56 (2000) 489–520.
[13] L.F. Landweber, L. Kari, The evolution of cellular computing: Nature's solution to a computational problem, in: Proceedings of the 4th DIMACS Meeting on DNA-Based Computers, Philadelphia, PA, 1998, pp. 3–15.
[14] L.F. Landweber, L. Kari, Universal molecular computation in ciliates, in: L.F. Landweber, E. Winfree (Eds.), Evolution as Computation, Springer, Berlin, Heidelberg, New York, 2002.
[15] I. Petre, S. Skogman, Gene assembly simulator, 2006. http://combio.abo.fi/simulator/simulator.php.
[16] D.M. Prescott, The evolutionary scrambling and developmental unscrambling of germline genes in hypotrichous ciliates, Nucl. Acids Res. 27 (1999) 1243–1250.
[17] D.M. Prescott, A. Ehrenfeucht, G. Rozenberg, Molecular operations for DNA processing in hypotrichous ciliates, Europ. J. Protistology 37 (2001) 241–260.